제 1 부 — 기초

4장
연산자와 식

From Symbols to Values

산술·관계·논리 연산자 · 우선순위 · 형 변환 · 수학 내장 함수

THE QUESTION

```
7 / 2      = 3
7.0 / 2.0  = 3.5

2 ** 3 ** 2 = 512
-2 ** 2     = -4

0.1 + 0.2 /= 0.3
```

학습 목표

이 장에서 익히는 네 가지

01

연산자 다루기

산술·관계·논리 연산자로 식을 구성하고, 각 연산이 어떤 자료형을 돌려주는지 안다.

02

우선순위 읽기

연산자 우선순위와 결합 방향을 알고, 괄호로 의도를 분명히 드러낸다.

03

형 변환 이해

혼합형 연산에서 형이 어떻게 변환되는지 파악하고 정수 나눗셈의 함정을 피한다.

04

수학 함수 활용

표준 수학 내장 함수를 식에 끌어 써서 공식을 그대로 코드로 옮긴다.

도입 문제

이 장을 마치면 풀 수 있다

1

이차방정식의 두 실근

$ax^2 + bx + c = 0$ 의 근의 공식을 코드로 옮긴다. 거듭제곱·제곱근·괄호가 얽힌 연산의 우선순위를 정확히 제어해야 한다.

```
x = (-b ± sqrt(b**2 - 4*a*c)) / (2*a)
```

2

7점을 둘이 나누면

7 / 2 라고 적으면 기대하던 3.5가 아니라 고장 난 듯한 3이 나온다. 그 이유와, 정확히 3.5를 얻는 방법을 규명한다.

```
print *, 7 / 2      →   3
```

두 문제의 핵심은 같다 — 컴퓨터 내부에서 어떤 연산이 먼저 일어나는가, 그리고 연산에 참여하는 데이터의 자료형이 무엇인가.

4.1 산술 연산자

다섯 개의 산술 연산자

`+`	덧셈	`3 + 4`	→	**7**
`-`	뺄셈	`3 - 4`	→	**-1**
`*`	곱셈	`3 * 4`	→	**12**
`/`	나눗셈	`7.0 / 2.0`	→	**3.5**
`**`	거듭제곱	`2 ** 10`	→	**1024**

나머지 연산자는 없다

나머지를 구하는 기호 연산자가 따로 없다. 대신 내장 함수 mod 와 modulo 를 쓴다.

정수끼리의 / 는 몫만 남긴다

소수부를 반올림 없이 버린다(truncation). 이 장 전체를 관통하는 가장 중요한 규칙이다.

```
7 / 2  =  3
```

4.1 계산 순서

수학의 사칙연산 법칙과 거의 같다

1	**괄호 ( )**	우선순위와 무관하게 무조건 최우선으로 계산된다. 복잡한 식일수록 괄호로 의도를 명시한다.
2	**거듭제곱 ****	사칙연산보다 먼저 계산된다. a + b**2 는 a + (b²) 이지 (a+b)² 가 아니다.
3	**곱셈·나눗셈 * /**	그다음 순위. 동등한 순위끼리는 왼쪽에서 오른쪽으로 계산된다.
4	**덧셈·뺄셈 + -**	가장 낮은 순위. 이 역시 왼쪽에서 오른쪽으로 계산된다.

[Fortran만의 규칙] 거듭제곱은 오른쪽 결합 — 2 ** 3 ** 2 는 $(2^3)^2 = 64$ 가 아니라 $2^{(3^2)} = 2^9 = 512$ 다.

4.1 [예제]

정수 나눗셈의 함정

```
integer :: a, b
real    :: x, y

a = 17;    b = 5
print *, "a / b     =", a / b       ! 정수 나눗셈 -> 3
print *, "mod(a, b) =", mod(a, b)   ! 나머지     -> 2
print *, "a ** 2    =", a ** 2

x = 17.0;  y = 5.0
print *, "x / y     =", x / y       ! 실수 나눗셈 -> 3.4
```

```
a / b     =           3
mod(a, b) =           2
x / y     =   3.40000010
```

정수형 나눗셈

17 / 5 는 결과도 정수로 보존된다. 수학적 결과 3.4 에서 소수부를 반올림 없이 버려 3이 된다.

실수형 나눗셈

17.0 / 5.0 은 소수점이 온전히 보존되어 3.4가 계산 된다.

왜 3.4 가 아니라 3.40000010 인가

고장이 아니다. 기본 단정밀도 real 의 유효숫자 약 7자리 한계에서 오는 근사값 현상이다.

4.1 관계 연산자

결과는 언제나 logical 이다

현대 표기	옛 표기	
`<`	`.lt.`	작다
`<=`	`.le.`	작거나 같다
`>`	`.gt.`	크다
`>=`	`.ge.`	크거나 같다
`==`	`.eq.`	같다
`/=`	`.ne.`	같지 않다

기호 표기만 쓴다

현대 Fortran 코딩에서는 < 나 == 같은 기호 표기만 전적으로 사용한다.

옛 표기도 알아 두자

.lt. · .eq. 같은 형태는 수십 년 된 방대한 수치 해석 레거시 코드에 여전히 남아 있다.

!= 가 아니라 /=

C나 파이썬은 '같지 않다'를 != 로 쓰지만 Fortran은 /= 를 쓴다.

4.1 논리 연산자

양옆에 반드시 마침표를 붙인다

p = .true. , q = .false. 일 때

`.not.`	**부정 (NOT)**	상태를 반대로 뒤집는다	`.not. p`	→ **.false.**
`.and.`	**논리곱 (AND)**	둘 다 참이어야 참이다	`p .and. q`	→ **.false.**
`.or.`	**논리합 (OR)**	둘 중 하나만 참이어도 참이다	`p .or. q`	→ **.true.**
`.eqv.`	**동치 (EQV)**	두 논리값이 서로 같으면 참이다	`p .eqv. q`	→ **.false.**
`.neqv.`	**배타적 논리합 (XOR)**	두 논리값이 서로 다르면 참이다	`p .neqv. q`	→ **.true.**

논리 연산자는 관계 연산자와 달리 양옆에 마침표(.)를 붙여야 컴파일러가 올바르게 인식한다.

4.1 [예제]

관계 연산과 논리 연산의 결합

```
integer :: age
logical :: is_adult, has_ticket, can_enter

age = 20
has_ticket = .true.

is_adult  = age >= 19            ! 관계식을 그대로 대입
can_enter = is_adult .and. has_ticket
```

```
is_adult  = T        can_enter = T
20 == 20  : T        3 /= 5    : T
not adult : F
```

logical 값은 화면에 출력될 때 마침표 없이 T 또는 F 라는 단일 문자로 표시된다.

관계식은 그 자체가 논리값이다

입문자는 참·거짓을 넣을 때 if 구문부터 떠올린다. 그러나 age >= 19 라는 관계식 자체가 이미 하나의 완전한 논리값이므로, 별도의 분기 없이 논리형 변수에 곧바로 대입할 수 있다.

```
if (age >= 19) then
   is_adult = .true.
```

↓ if 블록 없이 한 줄로

```
is_adult = (age >= 19)
```

4.2 연산자 우선순위

무엇이 먼저 계산되는가

높음

1	`**`	오른쪽 → 왼쪽
2	`*` `/`	왼쪽 → 오른쪽
3	단항 `+` `-` , 이항 `+` `-`	왼쪽 → 오른쪽
4	관계 연산자 `<` `<=` `>` `>=` `==` `/=`	—
5	`.not.`	—
6	`.and.`	왼쪽 → 오른쪽
7	`.or.`	왼쪽 → 오른쪽
8	`.eqv.` `.neqv.`	왼쪽 → 오른쪽

낮음

모든 산술 연산이 끝난 뒤에 관계 연산이 일어나고, 비교가 끝난 뒤에야 논리 연산이 수행된다.

4.2 [예제]

결합 방향이 결과를 바꾼다

```
2 + 3 * 4
```
= **14** 곱셈이 먼저

```
2 ** 3 ** 2
```
= **512** 오른쪽부터 — 2 ** (3 ** 2)

```
-2 ** 2
```
= **-4** 거듭제곱이 먼저 — -(2 ** 2)

```
10.0 / 2.0 / 5.0
```
= **1.0** 왼쪽부터 — (10 / 2) / 5

두 식은 같은 값을 내지만, 두 번째가 의도를 또렷이 보여 준다.

[흔한 실수] 거듭제곱의 방향

수학에서 $-2^2 = -4$ 인 것처럼 Fortran에서도 -2**2 는 -4 다. 음수 자체를 제곱하려면 반드시 (-2)**2 로 괄호를 씌운다.

괄호로 의도를 적는다

```
y = a + b * c
y = a + (b * c)
```

4.3 형 변환

혼합형(mixed-mode) 연산의 규칙

자동 변환은 식 전체가 아니라 연산자 단위로 일어난다

정수와 실수가 섞이면 Fortran은 더 넓은 표현 범위인 실수 쪽으로 정수를 올려 변환한 뒤 계산한다. 다만 이 변환은 식 전체를 보고 한 번에 일어나지 않고, 연산자 좌우의 피연산자만 보고 단계적으로 일어난다 — 그래서 식의 일부분이 정수끼리 먼저 계산되는 함정이 생긴다.

`real(i)`	정수 → 실수	`real(5)  →  5.0`
`int(x)`	실수 → 정수, 0 방향으로 버림	`int(3.9)  →  3`
`nint(x)`	실수 → 가장 가까운 정수, 반올림	`nint(3.9)  →  4`
`cmplx(a, b)`	두 실수 → 복소수	`cmplx(1.0, 2.0)  →  (1.0, 2.0)`

int 와 nint 는 다르다

int(4.5) 는 4 이지만 nint(4.5) 는 5 다. 목적에 맞는 함수를 정확히 골라야 한다.

정밀도는 kind 로 지정

real(total) 만 쓰면 단정밀도로 변환된다. real(total, real64) 처럼 종류를 항상 명시한다.

4.3 [예제]

도입 문제 2의 답 — 7 / 2 를 3.5 로

```
integer      :: total, count
real         :: avg_wrong
real(real64) :: avg_right

total = 7;  count = 2

avg_wrong = total / count                              ! 3.0
avg_right = real(total, real64) / real(count, real64) ! 3.5
```

```
avg_wrong          =    3.00000000
```

```
avg_right          =    3.5000000000000000
```

나누기 연산이 수행되기 전 단계에 형 변환 함수를 각각 적용한다.

왜 3.0 이 되는가

컴퓨터는 대입될 변수가 실수형이라는 사실을 전혀 고려하지 않는다. 오직 우변의 total / count 만 본다. 정수 7 ÷ 정수 2 = 정수 3 이 먼저 확정된 뒤 실수 방에 들어간다.

[흔한 실수] real(7 / 2)

3.5 가 아니라 3.0 이다. 괄호 안의 정수 나눗셈이 먼저 끝나기 때문이다. 나누기 전에 real(7) / real(2) 로 각각 변환해야 한다.

4.4 수학 내장 함수

외부 라이브러리 없이 바로 쓴다

거듭제곱 · 로그

`sqrt(x)` 제곱근
`exp(x)` 지수 함수 `e^x`
`log(x)` 자연로그
`log10(x)` 상용로그

삼각 · 역삼각

`sin(x)` `cos(x)` `tan(x)`
`asin(x)` `acos(x)` `atan(x)`
`atan2(y, x)` 사분면 고려
인수 단위는 라디안

절댓값 · 부호 · 나머지

`abs(x)` 절댓값
`sign(a, b)` `a` 크기에 `b` 부호
`mod(a, p)` 나머지
`modulo(a, p)` 나머지

최대 · 최소 · 올림 · 버림

`max(a, b, ...)` 최댓값
`min(a, b, ...)` 최솟값
`floor(x)` / `ceiling(x)`
`aint(x)` / `anint(x)`

대부분의 내장 함수는 포괄(generic) 함수다 — 단정밀도를 넣으면 단정밀도가, 배정밀도를 넣으면 배정밀도가 그대로 돌아온다.

4.4 [예제]

도입 문제 1의 답 — 근의 공식

$x^2 - 3x + 2 = 0$ 의 두 근을 $x = (-b \pm \sqrt{(b^2 - 4ac)}) / 2a$ 로 구한다

```
real(real64) :: a, b, c, disc, x1, x2

a =  1.0_real64;  b = -3.0_real64;  c = 2.0_real64

disc = b**2 - 4.0_real64 * a * c
x1 = (-b + sqrt(disc)) / (2.0_real64 * a)
x2 = (-b - sqrt(disc)) / (2.0_real64 * a)
```

```
discriminant = 1.0000000000000000     x1 = 2.0     x2 = 1.0
```

*거듭제곱 ** 은 곱셈보다 먼저 계산되므로 b**2 - 4.0*a*c 에는 괄호가 필요 없다.*

분모를 통째로 괄호로

괄호가 없으면 / 와 * 는 우선순위가 같아 왼쪽부터 계산된다. 분자를 2.0 으로 먼저 나눈 뒤 a 를 다시 곱하는, 수학적으로 완전히 틀린 식이 된다.

리터럴에도 _real64 를

식 안의 4.0 · 2.0 같은 상수에도 종류를 붙여야 연산 과정에서 정밀도가 떨어지지 않는다.

4.4 [예제]

mod 와 modulo 는 음수에서 갈린다

mod(a, p)

결과의 부호가 피제수 a 를 따른다

```
mod(7, 3)    =   1
mod(-7, 3)   =  -1
```

modulo(a, p)

결과의 부호가 제수 p 를 따른다

```
modulo(7, 3)    =  1
modulo(-7, 3)   =  2
```

양수끼리는 같다

두 인수가 모두 양수일 때는 계산 결과가 서로 일치한다.

언제 modulo 를 쓰나

시계 방향 순환이나 격자 배열의 주기적 인덱스처럼 결과를 항상 0 ~ p−1 범위로 감싸야 할 때 쓴다.

시뮬레이션의 선택

주기성을 가지는 격자 모델이나 시뮬레이션 인덱스 계산에서는 일반적으로 modulo 가 적합하다.

4.4 [흔한 실수]

삼각함수의 인수는 라디안이다

sin(90.0) 은 1.0 이 아니다

Fortran의 모든 삼각함수는 '도(degree)'가 아니라 '라디안(radian)' 단위로 작동한다. sin(90.0) 이라고 쓰면 sin(90°) = 1.0 이 아니라 90 라디안에 대한 사인 값이 계산된다.

각도를 쓰려면 변환식을 연산 안에 직접 포함한다

```
real(real64), parameter :: pi = 3.141592653589793_real64
real(real64) :: result

result = sin(90.0_real64 * pi / 180.0_real64)
```

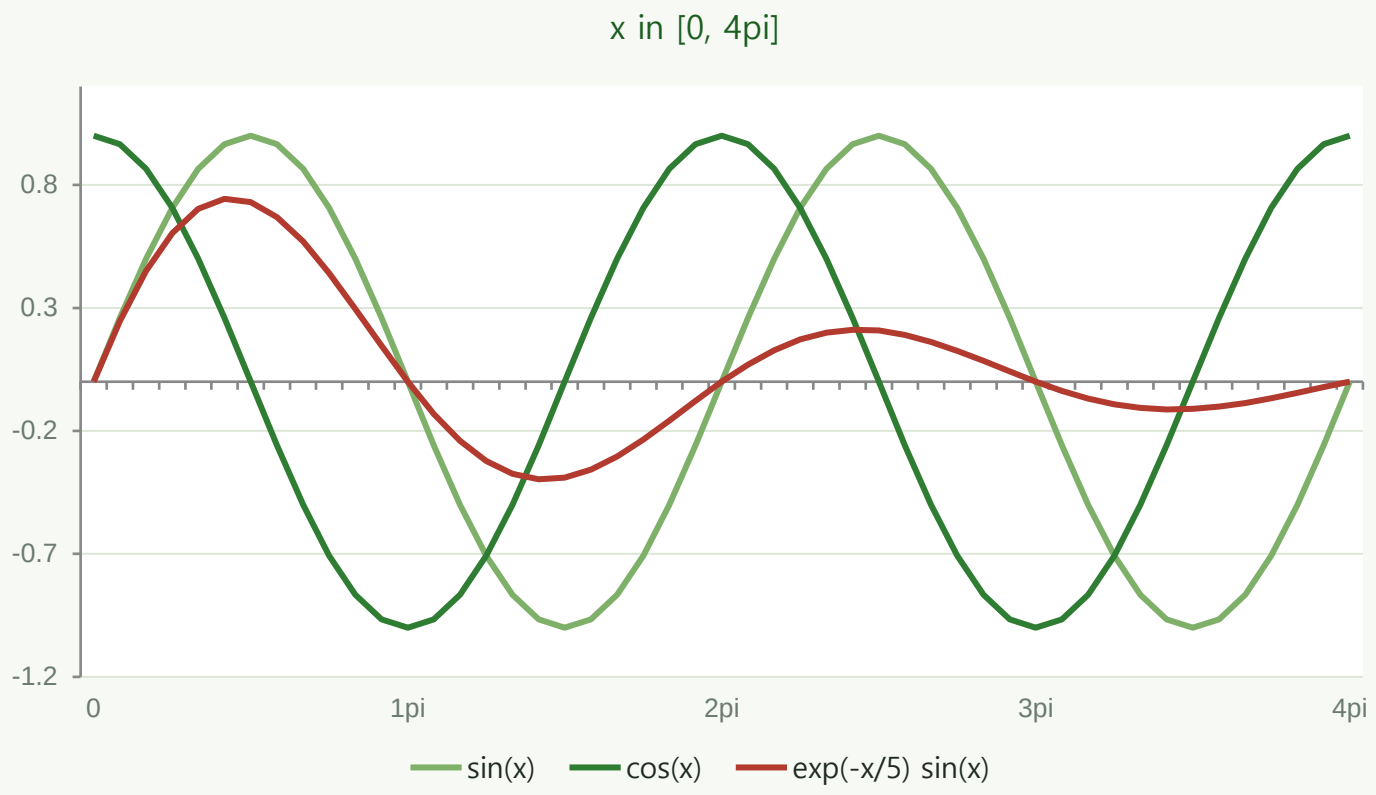
변환 상수는 parameter 로

π/180 을 명명 상수로 선언해 두면 식이 짧아지고, 값이 코드 곳곳에 흩어지는 매직 넘버를 막을 수 있다.

역삼각함수 asin · acos · atan 이 돌려주는 값도 라디안이다. atan2(y, x) 는 점 (x, y)가 놓인 사분면까지 고려해 올바른 각도를 준다.

4.4 [예제]

Fortran이 계산하고 Python이 겹쳐 그린다




```
x  = real(i,real64)/real(n,real64) &
     * 4.0_real64 * pi
f1 = sin(x)
f2 = cos(x)
f3 = exp(-x / 5.0_real64) * sin(x)
```

기본 파동

sin 과 cos 는 같은 진폭 1.0 을 유지한 채 정확히 90°(π/2) 위상 차를 두고 맞물려 흐른다.

감쇠 파동

지수 감쇠 성분이 곱해져 x 가 커질수록 진폭이 점진적으로 잦아든다.

산술 연산 · 거듭제곱 · 수학 함수를 모두 써서 격자점 위의 함숫값을 생성하고 csv로 내보낸 뒤 파이썬으로 그렸다.

오류 학습

왜 0.1 + 0.2 는 0.3 이 아닌가

컴퓨터는 모든 숫자를 2진법으로 바꾸어 저장한다. 2진수는 분모가 2의 거듭제곱인 분수만 유한소수로 깨끗하게 떨어진다.

0.5	= 1/2	0.1 (2진수)	정확하게 표현된다
0.1	= 1/10	0.0001100110011001... (2진수)	0011 이 무한히 반복된다

메모리는 32비트든 64비트든 항상 유한하다. 끝없이 이어지는 무한소수를 다 담을 수 없으므로, 컴퓨터는 가장 가까운 근사값에서 꼬리를 잘라 저장한다.

0.1_real64 + 0.2_real64 = 0.30000000000000004 ← IEEE 754 를 따르는 모든 언어(C, C++, Python, Java)의 공통 한계다.

오류 학습

실수를 == 로 비교하지 않는다

매우 위험한 코드

```
if (x == 0.3_dp) then
   ! x 가 0.30000000000000004 라면
   ! 이 조건은 거짓이 된다
```

미세한 근사값 오차 때문에, 논리적으로 참이어야 할 조건문이 거짓으로 판정되어 블록이 실행되지 않을 수 있다.

올바르고 안전한 코드

```
if (abs(x - 0.3_dp) < 1.0e-12_dp) then
   ! 차이가 허용 오차보다 작으면
   ! 사실상 '같다'고 판정한다
```

두 값의 차이를 뺀 뒤 절댓값을 취해, 아주 미세한 허용 오차(에프실론)보다 작은지를 검증하는 방식으로 설계한다.

컴파일러도 경고해 준다 — -Wall -Wextra

```
Warning: Equality comparison for REAL(8) at (1) [-Wcompare-reals]
 x = 0.30000000000000004      x == 0.3 : F      abs diff : 5.5511151231257827E-017
```

오류 학습

컴파일러는 훌륭한 감시자다

옵션을 켜 두면 번역 과정에서 인간이 미처 발견하지 못한 자료형의 모순이나 잠재적 버그를 찾아낸다.

[에러] 컴파일 단계에서 차단

```
integer :: n
n = 16
r = sqrt(n)      ! 정수를 넣을 수 없다
```

```
Error: 'x' argument of 'sqrt' intrinsic
  at (1) must be REAL or COMPLEX
```

제곱근은 연속적인 실수를 다루는 연산이다. r = sqrt(real(n)) 처럼 형 변환을 거쳐야 한다.

[경고] 통과하지만 런타임 버그

```
x = 0.1_real64 + 0.2_real64
print *, "x == 0.3 :", x == 0.3_real64
```

```
Warning: Equality comparison for REAL(8)
  at (1) [-Wcompare-reals]
```

경고는 컴파일을 막지 않는다. "이 비교문은 실행할 때 정상 작동하지 않을 확률이 높다"는 신호다.

4장 정리

요약

연산자와 우선순위

- 산술 연산자는 + − * / ** 다섯 개. 나머지 연산자는 없고 mod / modulo 함수를 쓴다.
- 관계 연산자와 논리 연산자의 결과는 언제나 logical 이다.
- 우선순위: ** → * / → + − → 관계 → .not. → .and. → .or. → .eqv./.neqv.
- ** 는 오른쪽 결합이며(2**3**2 = 512), -2**2 는 -(2**2) = -4 다.
- mod 는 피제수의 부호를, modulo 는 제수의 부호를 따른다.

형 변환과 함수

- 정수끼리의 / 는 몫만 남긴다. 실수 결과가 필요하면 나누기 전에 변환한다.
- 변환 함수는 real(→실수) · int(버림) · nint(반올림) · cmplx(→복소수).
- 정밀도는 real*8 이 아니라 iso_fortran_env 의 real64 와 kind 로 지정한다.
- 삼각함수의 인수는 라디안이다.
- 실수는 == 로 비교하지 않고, 차이의 절댓값을 허용 오차와 비교한다.

한 줄 요약: 무엇이 먼저 계산되는지, 그리고 연산에 참여하는 자료형이 무엇인지를 항상 의식한다.

연습 문제

1. a = 23, b = 4 에 대해 a+b, a−b, a*b, a/b, mod(a,b), a**2 를 각각 출력하라.

2. 7 과 2 의 정수 몫과 배정밀도 실수 나눗셈 결과를 출력하고 차이를 주석으로 설명하라.

3. 2+3*4, (2+3)*(4−5), 2**2**3, −3**2 의 값을 손으로 예측한 뒤 프로그램으로 확인하라.

4. "65세 이상이거나 회원이면 할인 대상"을 하나의 논리식으로 만들어 결과를 출력하라.

5. int(2.9), int(−2.9), nint(2.5), nint(−2.5), floor(−2.1), ceiling(−2.1) 을 표로 정리하라.

6. mod(−9, 4) 와 modulo(−9, 4) 를 출력하고 부호가 다른 이유를 한 문장으로 설명하라.

7. 섭씨 c 를 화씨로 변환해 출력하라. 정수 나눗셈 함정을 피하도록 모든 상수를 실수로 적을 것.

8. 두 변 a=3, b=4 (배정밀도)로 빗변 sqrt(a**2 + b**2) 의 길이를 구하라.

9. 각도 deg 를 라디안으로 바꿔 sin·cos·tan 을 출력하라. π/180 은 parameter 로 선언한다.

10. 원금 p, 연이율 r, 기간 t 로 복리 원리합계 p(1+r)^t 를 배정밀도로 계산하라.

다음 장에서는 제어 구조 — if 조건문과 do 반복문을 다룬다.